1

From Domain to Requirements*

Dines Bjørner

[1] Faculté des Sciences, Bureau 266, LORIA & Université Henri Poincaré Nancy 1, BP 239, F-54506 Vandœuvre lès Nancy, France.**
[2] Professor emeritus, DTU Informatics, Bldg. 325, Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark.
[3] Fredsvej 11, DK-2840 Holte, Danmark.
E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Abstract We first present a summary of essentials of domain engineering, its motivation, and its modelling of abstractions of domains through the modelling of the intrinsics, support technologies, management and organisation, rules and regulations, scripts, and human behaviour of whichever domain is being described.

Then we present the essence of two (of three) aspects of requirements: the domain requirements and the interface requirements prescriptions as they relate to domain descriptions and we survey the basic operations that "turn" a domain description into a domain requirements prescription: projection, instantiation, determination, extension and fitting. An essence of interface requirements is also presented: the "merging" of shared entities, operations, events and behaviours of the domain with those of the machine (i.e., the hardware and software to be designed).

1.1 Introduction

This paper presents a model of early stages of software development that is not conventional. The model is presented in two alternating ways: (i) we present some of the principles and techniques of that unconventional software development method, and (ii) we present — what in the end, that is, taken across the paper, amounts to a relatively large example.

In summary: the objective of the present paper is to relate domain engineering to requirements engineering and to show that one can obtain an altogether different basis for requirements engineering.

* Invited paper for the Festschrift for Ugo Montanari on the occasion of his 65th anniversary June 12, 2008. Edited by Rocco di Nicola et al.

** This paper was written with the financial support of Université Henri Poincaré and INRIA during the author's two month visit: October 15 – December 14, 2007.

1.2 The Triptych Principle of Software Engineering

We start, unconventionally, by enunciating a principle. The principle expresses how we see software development as centrally consisting of three "programming-like" phases based on the following observation: before software can be designed we must understand its requirements, and before requirements can be prescribed we must understand the application domain. We therefore see software development proceeding, ideally, in three phases: a first phase of domain engineering, a second phase of requirements engineering, and a third phase of software design.

The first paragraphs of Sects. 1.3 and 1.4 explain what the objectives of domain engineering and requirements engineering are. The sections otherwise outline major development stages and steps of these two phases.

1.3 Domain Engineering

The objective of domain engineering is to create a domain description. A domain description specifies entities, functions, events and behaviours of the domain such as the domain stakeholders think they are. A domain description thus (indicatively) expresses what there is. A domain description expresses no requirements let alone anything about the possibly desired (required) software.

1.3.1 Stages of Domain Engineering

To develop a proper domain description necessitates a number of development stages: (i) identification of stakeholders, (ii) domain knowledge acquisition, (iii) business process rough-sketching, (iv) domain analysis, (v) domain modelling: developing abstractions and verifying properties, (vi) domain validation and (vii) domain theory building.

Business process (BP) rough-sketching amount to rough, narrative outlines of the set of business processes as experienced by each of the stakeholder groups. BP engineering is in contrast to BR re-engineering (BPR) which we shall cover later, but briefly in Sect. 1.4.2.

We shall only cover domain modelling.

1.3.2 First Example of a Domain Description

We exemplify a transportation domain. By transportation we shall mean *the movement of vehicles from hubs to hubs along the links of a net.*

Rough Sketching — Business Processes

The basic *entities* of the transportation "business" are the (i) *net*s with their (ii) *hub*s and (iii) *link*s, the (iv) *vehicle*s, and the (v) *traffic* (of vehicles on the net). The basic *functions* are those of (vi) vehicles entering and leaving the net (here simplified to entering and leaving at hubs), (vii) for vehicles to make movement transitions along the net, and (viii) for inserting and removing links (and associated hubs) into and from the net. The basic *events* are those of (ix) the appearance and disappearance of vehicles, and (x) the breakdown of links. And, finally, the basic behaviours of the transportation business are those of (xi) vehicle journey through the net and (xii) net development & maintenance including insertion into and removal from the net of links (and hubs).

Narrative — Entities

By an *entity* we mean *something we can point to, i.e., something manifest, or a concept abstracted from, such a phenomenon or concept thereof.*

Among the many entities of transportation we start with nets, hubs, and links.

A transportation net consists of hubs and links. Hubs and links are different kinds of entities. Conceptually hubs (links) can be uniquely identified. From a link one can observe the identities of the two distinct hubs it links. From a hub one can observe the identities of the one or more distinct links it connects.

Other entities such as vehicles and traffic could as well be described. Please think of these descriptions of entities as descriptions of the real phenomena and (at least postulated) concepts of an actual domain.

Formalisation — Entities

```
type H, HI, L, LI, N = H-set × L-set
value obs_HI: H→HI, obs_LI: L→LI, obs_HIs: L→HI-set,obs_LIs: H→LI-set
axiom
  ∀ (hs,ls):N •
    card hs≥2 ∧ card ls≥1 ∧ ∀ h:H • h ∈ hs ⇒
      ∀ li:LI • li ∈ obs_LIs(h) ⇒
        ∃ l′:L • l′ ∈ ls∧li=obs_LI()∧obs_HI(h) ∈ obs_HIs(l′)∧
    ∀ l:L • l ∈ ls ⇒
        ∃ h′,h″:H • {h′,h″}⊆hs∧obs_HIs(l)={obs_HI(h′),obs_HI(h″)}
value xtr_HIs: N → HI-set,xtr_LIs: N → LI-set
```

Narrative — Operations

By an *operation* (of a domain) we mean *a function that applies to entities of the domain and yield entities of that domain — whether these entities are actual phenomena or concepts of these or of other phenomena.*

Actions (by domain stakeholders) amount to the execution of operations.

Among the many operations performed in connection with transportation we illustrate some on nets. To a net one can join new links in either of three ways: The new link connects two new hubs — so these must also be joined , or The new link connects a new hub with an existing hub — so it must also be joined, or The new link connects two existing hubs. In any case we must either provide the new hubs or identify the existing hubs.

From a net one can remove a link. Three possibilities now exists: The removed link would leave its two connected hubs isolated unless they are also removed — so they are; The removed link would leave one of its connected hubs isolated unless it is also removed — so it is; or The removed link connects two hubs into both of which other links are connected — so all is OK. (Note our concern for net invariance.) Please think of these descriptions of operations as descriptions of the real phenomena and (at least postulated) concepts of an actual domain. (Thus they are not prescriptions of requirements to software let alone specifications of software operations.)

Formalisation — Operations

```
type
  NetOp = InsLnk | RemLnk
  InsLnk == 2Hs(h1:H,l:L,h2:H)|1H(hi:HI,l:L,h:H)|0H(hi1:HI,l:L,hi2:HI)
  RemLnk == RmvL(li:LI)
value
  int_NetOp: NetOp → N ~→ N
  pre int_NetOp(op)(hs,ls) ≡
    case op of
      2Hs(h1,l,h2) →
        {h1,h2}∩ hs={}∧l∉ ls∧
        obs_HIs(l)={obs_HI(h1),obs_HI(h2)}∧
          {obs_HI(h1),obs_HI(h2)}∩ xtr_HIs(hs)={}∧
          obs_LIs(h1)={li}∧obs_LIs(h2)={li},
      1H(hi,l,h) → ...,
      0H(hi1,l,hi2) → ...
    end


  int_NetOp(op)(hs,ls) ≡
    case op of
```

```
        2Hs(h1,l,h2) →
            (hs ∪ {h1,h2},ls ∪ {l}),
        1H(hi,l,h) →
            (hs\{xtr_H(hi,hs)}∪{h,aLI(xtr_H(hi,hs),obs_LI(l))},ls ∪ {l}),
        0H(hi1,l,hi2) → ...,
        RmvL(li) → ...
    end

  xtr_H: HI × H-set ~→ H
  xtr_H(hi,hs) ≡ let h:H • h ∈ hs ∧ obs_HI(h)=hi in h end
  pre ∃ h:H • h ∈ hs ∧ obs_HI(h)=hi


  aLI: H × LI → H, sLI: H × LI → H
  aLI(h,li) as h′,
    pre li ∉ obs_LIs(h), post obs_LIs(h′)={li} ∪ obs_LIs(h)∧ ...
  sLI(h′,li) as h,
    pre li ∈ obs_LIs(h′), post obs_LIs(h)=obs_LIs(h′)\{li}∧ ...
```

The ellipses, . . . , shall indicate that previous properties of h holds for h′.

Narrative — Events

By an *event* of a domain we shall here mean *an instantaneous change of domain state (here, for example, "the" net state) not directly brought about by some willed action of the domain but either by "external" forces or implicitly, as an unintended result of a willed action.*

Among the "zillions" of events that may occur in transportation we single out just one. A link of a net ceases to exist as a link.[4]

In order to model transportation events we — ad hoc — introduce a transportation state notion of a net paired with some — ad hoc — "conglomerate" of remaining state concepts referred to as $\omega : \Omega$.

Formalisation — Events

```
type
  Link_Disruption == LiDi(li:LI)
channel
  x:(Link_Disruption|...)
value
  transportation_transition: (N × Ω) → in x  (N × Ω)
  transportation_transition(n,ω) ≡
    ... ⫿ let xv = x? in
```

```
                case xv of
                    LiDi(li) → (int_NetOp(RmvL(li))(hs,ls),line_dis(ω))
                    ... end end ⌈⌉ ...
  line_dis: Ω → Ω
```

Narrative — Behaviours

By a *behaviour* we mean *a possibly infinite sequence of zero, one or more actions and events.*

We illustrate just one of very many possible transportation behaviours.

A net behaviour is a sequence of zero, one or more executed net operations: the openings (insertions) of new links (and implied hubs) and the closing (removals) of existing links (and implied hubs), and occurrences of external events (limited here to link disruptions).

Formalisation — Behaviours

```
channel
  x:...
value
  transportation_transition: (N × Ω) → in x  (N × Ω)
  transportation_transition(n,ω) ≡
    ... ⌈⌉ let xv = x? in case xv of ... end end
    ... ⌈⌉ let op:NetOp • pre IntNetOp(op)(n) in IntNetOp(op)(n) end ...
  transportation: (N × Ω) → in x  Unit
  transportation(n,ω) ≡
    let (n′,ω′) = transportation_transition(n,ω) in
    transportation (n′,ω′) end
```

1.3.3 Domain Modelling: Describing Facets

Domain modelling, as we shall see, entails modelling a number of domain facets.

By a *domain facet* we mean *one amongst a finite set of generic ways of analysing a domain: a view of the domain, such that the different facets cover conceptually different views, and such that these views together cover the domain.*

These are the facets that we find "span" a domain in a pragmatically sound way: intrinsics, support technology, management & organisation, rules & regulations, scripts and human behaviour: We shall now survey these facets.

Domain Intrinsics

By *domain intrinsics* we mean *those phenomena and concepts of a domain which are basic to any of the other facets (listed earlier and treated, in some detail, below), with such domain intrinsics initially covering at least one specific, hence named, stakeholder view.*

For the large example of Sect. 1.3.2, we claim that the net, hubs and links were intrinsic phenomena of the transportation domain; and that the operations of joining and removing links were not: one can explain transportation without these operations. We will now augment the domain description of Sect. 1.3.2 with an intrinsic concept, namely that of the states of hubs and links: where these states indicate desirable directions of flow of movement.

A Transportation Intrinsics — Narrative.

With a hub we can associate a concept of hub state. The pragmatics of a hub state is that it indicates desirable directions of flow of vehicle movement from (incoming) links to (outgoing) links. The syntax of indicating a hub state is (therefore) that of a possibly empty set of triples of two link identifiers and one hub identifier where the link identifiers are those observable from the identified hub.

With a link we can associate a concept of link state. The pragmatics of a link state is that it indicates desirable directions of flow of vehicle movement from (incoming, identified) hubs to (outgoing, identified) hubs along an identified link. The syntax of indicating a link state is (therefore) that of a possibly empty set of triples of pairs of identifiers of link connected hub and a link identifier where the hub identifiers are those observable from the identified link.

A Transportation Intrinsics — Formalisation.

```
type
  X = LI×HI×LI [ crossings of a hub ], P = HI×LI×HI [ paths of a link ]
  HΣ = X-set [ hub states ],              LΣ = P-set [ link states ]
value
  obs_HΣ: H → HΣ, obs_LΣ: L → LΣ,
  xtr_Xs: H → X-set, xtr_Ps: L → P-set
  xtr_Xs(h) ≡
    {(li,hi,li′)|li,li′:LI,hi:HI•{li,li′}⊆obs_LIs(h)∧hi=obs_HI(h)}
  xtr_Ps(l) ≡
    {(hi,li,hi′)|hi,hi′:HI,li:LI•{hi,hi′}=obs_HIs(l)∧li=obs_LI(l)}
axiom ∀ n:N,h:H;l:L • h ∈ obs_Hs(n)∧l ∈ obs_Ls(n) ⇒
        obs_HΣ(h)⊆xtr_Xs(h) ∧ obs_LΣ(l)⊆xtr_Ps(l)
```

Domain Support Technologies

By *domain support technologies* we mean *ways and means of implementing certain observed phenomena or certain conceived concepts.*

A Transportation Support Technology Facet — Narrative, 1.

Earlier we claimed that the concept of hub and link states was an intrinsics facet of transport nets. But we did not describe how hubs or links might change state, yet hub and link state changes should also be considered intrinsic facets. We there introduce the notions of hub and link state spaces and hub and link state changing operations. A hub (link) state space is the set of all states that the hub (link) may be in. A hub (link) state changing operation can be designated by the hub and a possibly new hub state (the link and a possibly new link state).

A Transportation Support Technology Facet — Formalisation, 1.

type HΩ = HΣ**-set**, LΩ = LΣ**-set**
value obs_HΩ: H → HΩ, obs_LΩ: L → LΩ
axiom ∀ h:H • obs_HΣ(h) ∈ obs_HΩ(h) ∧ ∀ l:L • obs_LΣ(l) ∈ obs_LΩ(l)
value
  chg_HΣ: H × HΣ → H, chg_LΣ: L × LΣ → L
  chg_HΣ(h,hσ) **as** h′, **pre** hσ ∈ obs_HΩ(h), **post** obs_HΣ(h′)=hσ
  chg_LΣ(l,lσ) **as** l′, **pre** lσ ∈ obs_LΩ(h), **post** obs_HΣ(l′)=lσ

A Transportation Support Technology Facet — Narrative, 2.

Well, so far we have indicated that there is an operation that can change hub and link states. But one may debate whether those operations shown are really examples of a support technology. (That is, one could equally well claim that they remain examples of intrinsic facets.) We may accept that and then ask the question: How to effect the described state changing functions ? In a simple street crossing a semaphore does not instantaneously change from red to green in one direction while changing from green to red in the cross direction. Rather there is are intermediate sequences of green/yellow/red and red/yellow/green states to help avoid vehicle crashes and to prepare vehicle drivers. Our "solution" is to modify the hub state notion.

A Transportation Support Technology Facet — Formalisation, 2.

type
  Colour == red | yellow | green
  X = LI×HI×LI×Colour [crossings **of** a hub]
  HΣ = X**-set** [hub states]

```
value
  obs_HΣ: H → HΣ, xtr_Xs: H → X-set
  xtr_Xs(h) ≡
    {(li,hi,li′,c)|li,li′:LI,hi:HI,c:Colour•{li,li′}⊆obs_LIs(h)∧hi=obs_HI(h)}
axiom
  ∀ n:N,h:H • h ∈ obs_Hs(n) ⇒ obs_HΣ(h)⊆xtr_Xs(h) ∧
    ∀ (li1,hi2,li3,c),(li4,hi5,li6,c′):X •
      {(li1,hi2,li3,c),(li4,hi5,li6,c′)}⊆obs_HΣ(h) ∧
      li1=li4 ∧ hi2=hi5 ∧ li3=li6 ⇒ c=c′
```

A Transportation Support Technology Facet — Narrative, 3.

We consider the colouring, or any such scheme, an aspect of a support technology facet. There remains, however, a description of how the technology that supports the intermediate sequences of colour changing hub states.

We can think of each hub being provided with a mapping from pairs of "stable" (that is non-yellow coloured) hub states ($h\sigma_i$,$h\sigma_f$) to well-ordered sequences of intermediate "un-stable' (that is yellow coloured) hub states paired with some time interval information $\langle(h\sigma', t\delta'), (h\sigma'', t\delta''), \ldots, (h\sigma'^{\cdots\prime}, t\delta'^{\cdots\prime})\rangle$ and so that each of these intermediate states can be set, according to the time interval information,[5] before the final hub state ($h\sigma_f$) is set.

A Transportation Support Technology Facet — Formalisation, 3.

```
type
  TI [ time interval ]
  Signalling = (HΣ × TI)*
  Sema = (HΣ × HΣ) m Signalling
value
  obs_Sema: H → Sema,
  chg_HΣ: H × HΣ → H,
  chg_HΣ_Seq: H × HΣ → H
  chg_HΣ(h,hσ) as h′
    pre hσ ∈ obs_HΩ(h) post obs_HΣ(h′)=hσ
  chg_HΣ_Seq(h,hσ) ≡
    let sigseq = (obs_Sema(h))(obs_Σ(h),hσ) in sig_seq(h)(sigseq) end
  sig_seq: H → Signalling → H
  sig_seq(h)(sigseq) ≡
    if sigseq=⟨⟩ then h else
    let (hσ,tδ) = hd sigseq in let h′ = chg_HΣ(h,hσ);
    wait tδ;
    sig_seq(h′)(tl sigseq) end end end
```

Domain Management & Organisation

By *domain management* we mean *people (such decisions) (i) who (which) determine, formulate and thus set standards (cf. rules and regulations, a later lecture topic) concerning strategic, tactical and operational decisions; (ii) who ensure that these decisions are passed on to (lower) levels of management, and to "floor" staff; (iii) who make sure that such orders, as they were, are indeed carried out; (iv) who handle undesirable deviations in the carrying out of these orders cum decisions; and (v) who "backstop" complaints from lower management levels and from floor staff.*

We use the connective '&' (ampersand) in lieu of the connective 'and' in order to emphasise that the joined concepts (A & B) hang so tightly together that it does not make sense to discuss one without discussing the other.

By *domain organisation* we mean *the structuring of management and non-management staff levels; the allocation of strategic, tactical and operational concerns to within management and non-management staff levels; and hence the "lines of command": who does what and who reports to whom — administratively and functionally.*

A Transportation Management & Organisation Facet — Narrative.

In the previous section on support technology we did not describe who or which "ordered" the change of hub states. We could claim that this might very well be a task for management.

(We here look aside from such possibilities that the domain being modelled has some further support technology which advices individual hub controllers as when to change signals and then into which states. We are interested in finding an example of a management & organisation facet — and the upcoming one might do!)

So we think of a 'net hub state management' for a given net. That management is divided into a number of 'sub-net hub state managements' where the sub-nets form a partitioning of the whole net. For each sub-net management there are two kinds management interfaces: one to the overall hub state management, and one for each of interfacing sub-nets. What these managements do, what traffic state information they monitor, etcetera, you can yourself "dream" up. Our point is this: We have identified a management organisation.

A Transportation Management & Organisation Facet — Formalisation.

type
  HIsLIs = HI-**set**×LI-**set**,
  MgtNet′ = HIsLIs×N, MgtNet={|mgtnet:MgtNet′•wf_MgtNet(mgtnet)|}
  Part′ = HIsLIs-**set**×N, Part={|part:Part′•wf_Part(part)|}
value

```
wf_MgtNet: MgtNet′ → Bool
wf_MgtNet((his,lis),n) ≡
  [ The his component contains all the hub ids.
    of links identified in lis ]
wf_Part: Part′ → Bool
wf_Part(hisliss,n) ≡
  ∀ (his,lis):HIsLIs •
    (his,lis) ∈ hisliss ⇒ wf_MgtNet((his,lis),n)∧
    [ no sub−net overlap and together they ″span″ n ]
```

Etcetera.

Domain Rules & Regulations

Domain Rules.

By a *domain rule* we mean *some text (in the domain) which prescribes how people or equipment are expected to behave when dispatching their duty, respectively when performing their function.*

Domain Regulations.

By a *domain regulation* we mean *some text (in the domain) which prescribes what remedial actions are to be taken when it is decided that a rule has not been followed according to its intention.*

A Transportation Rules & Regulations Facet — Narrative.

The purpose of maintaining an appropriate set of hub (and link) states may very well be to guide traffic into "smooth sailing" — avoiding traffic accidents etc. But this requires that vehicle drivers obey the hub states, that is, the signals. So there is undoubtedly a rule that says: *Obey traffic signals.*And, in consequence of human nature, overlooking or outright violating signals there is undoubtedly a regulation that says: *Violation of traffic signals is subject to fines and . . . .*

A Transportation Rules & Regulations Facet — Formalisation.

We shall, regretfully, not show any formalisation of the above mentioned rule and regulation. To do a proper job at such a formalisation would require that we formalise traffics, say as (a type of) continuous functions from time to pairs of net and vehicle positions, that we define a number of auxiliary (traffic monitoring) functions, including such which test whether from one instance of traffic, say at time t to a "next" instance of time, t', some one or more

vehicles have violated the rule[6], etc. The "etcetera" is ominous: It implies modelling traffic wardens (police trying to apprehend the "sinner"), 'etc.' ! We rough-sketch an incomplete formalisation.

```
type
   T [time], V [vehicle], Rel_Distance = {| f:Rel • 0<f<1 |}
   VPos == VatH(h:H) | VonL(hif:HI,l:L,hit:HI,rel_distance:Rel_Distance)
   Traffic = T → (N × (V m→ VPos))
value violations: Traffic → (T×T) → V-set
```

Vehicle positions are either at hubs or some fraction f down a link (l) from some hub (hit) towards the connected hub (hit). Traffic maps time into vehicle positions. We omit a lengthy description of traffic well-formedness.

Domain Scripts

By a *domain script* we mean *the structured, almost, if not outright, formally expressed, wording of a rule or a regulation that has legally binding power, that is, which may be contested in a court of law.*

A Transportation Script Facet — Narrative.

Regular buses ply the network according to some time table. We consider a train time table to be a script. Let us take the following to be a sufficiency narrative description of a train time table. For every train line, identified by a line number unique to within, say a year of operation, there is a list of train hub visits. A train hub visit informs of the intended arrival and departure times at identified hubs (i.e., train stations) such that "neighbouring" hub visits, (t_{a_i}, h_i, t_{d_i}) and (t_{a_j}, h_j, t_{d_j}), satisfy the obvious that a train cannot depart before it has arrived, and cannot arrive at the next, the "neighbouring" station before it has departed from the previous station, in fact, $t_{a_j} - t_{d_i}$ must be commensurate with the distance between the two stations.

A Transportation Script Facet — Formalisation.

```
type
   TLin
   HVis = T × HI × T
   Journey′ = HVis*, Journey = {|j:Journey′•len j≥2|}
   TimTbl′ = (TLin m→ Journey) × N
   TimTbl = {| timtbl:TimTbl′ • wf_TimTbl(timtbl) |}
value
   wf_TimTbl: TimTbl′ → Bool
   wf_TimTbl(tt,n) ≡
```

[all hubs designated in tt must be hubs of n]
[and all journeys must be along feasible links of n]
[and with commensurate timing net n constraints]

Domain Human Behaviour

By *human behaviour* we mean *any of a quality spectrum of carrying out assigned work: from (i)* ***careful, diligent*** *and* ***accurate,*** *via (ii)* ***sloppy*** *dispatch, and (iii)* ***delinquent*** *work, to (iv) outright* ***criminal*** *pursuit.*

Transportation Human Behaviour Facets — Narrative.

We have already exemplified aspects of human behaviour in the context of the transportation domain, namely vehicle drivers not obeying hub states. Other example can be given: drivers moving their vehicle along a link in a non-open direction, drivers waving their vehicle off and on the link, etcetera. Whether rules exists that may prohibit this is, perhaps, irrelevant. In any case we can "speak" of such driver behaviours — and then we ought formalise them !

Transportation Human Behaviour Facets — Formalisation.

But we decide not to. For the same reason that we skimped proper formalisation of the violation of the *"obey traffic signals"* rule. But, by now, you've seen enough formulas and you ought trust that it can be done.

off_on_link: Traffic → (T×T) $\xrightarrow{\sim}$ (V $\xrightarrow[m]{}$ VPos×VPos)
wrong_direction: Traffic → T $\xrightarrow{\sim}$ (V $\xrightarrow[m]{}$ VPos)

1.3.4 Discussion

We have given a mere glimpse of a domain description. A full description of a reasonably "convincing" domain description will take years to develop and will fill many pages (hundreds, . . . (!)).

1.4 Requirements Engineering

The objective of requirements engineering is to create a requirements prescription: A requirements prescription specifies externally observable properties of entities, functions, events and behaviours of *the machine* such as the requirements stakeholders wish them to be. The *machine* is what is required: that

is, the *hardware* and *software* that is to be designed and which are to satisfy the requirements. A *requirements prescription* thus (*putatively*) expresses what there should be. A requirements prescription expresses nothing about the design of the possibly desired (required) software. We shall show how a major part of a requirements prescription can be "derived" from "its" prerequisite domain description.

The Example Requirements

The domain was that of transportation. The requirements is now basically related to the issuance of tickets upon vehicle entry to a toll road net and payment of tickets upon the vehicle leaving the toll road net both issuance and collection/payment of tickets occurring at toll booths which are hubs somehow linked to the toll road net proper. Add to this that vehicle tickets are sensed and updated whenever the vehicle crosses an intermediate toll road intersection.

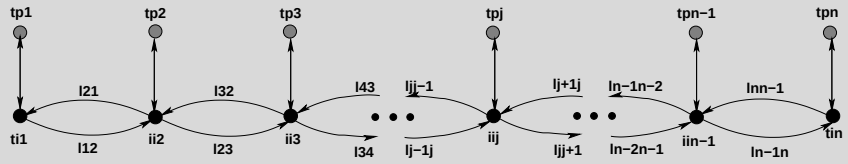

Fig. 1.1. A simple, linear toll road net: tp_i: *t*oll *p*laza i, ti_1, ti_n: *t*erminal *i*ntersection k, ii_k: *i*ntermediate *i*ntersection k, $1<k<n$ l_{xy}: tollway *l*ink from i_x to i_y, $y=x+1$ or $y=x\text{-}1$ and $1\leq x<n$.

1.4.1 Stages of Requirements Engineering

The following are the stages of requirements engineering: stakeholder identification, *business process re-engineering*, *domain requirements development*, *interface development*, machine requirements development, requirements verification and validation, and requirements satisfiability and feasibility.

The domain requirements development stage consists of a number of steps: projection, instantiation, determination, extension, and fitting.

We shall basically only cover business process re-engineering and domain requirements development

1.4.2 Business Process Re-engineering

Business process re-engineering (BPR) re-evaluates the intrinsics, support technologies, management & organisation, rules & regulations, scripts, and human behaviour facets while possibly changing some or all of these, that is, possibly rewriting the corresponding parts of the domain description.

Re-engineering Domain Entities

The net is arranged as a linear sequence of two or more (what we shall call) intersection hubs. Each intersection hub has a single two-way link to (what we shall call) an entry/exit hub (toll plaza); and each intersection hub has either two or four one-way (what we shall call) tollway links: the first and the last intersection hub (in the sequence) has two tollway links and all (what we shall call) intermediate intersections has four tollway links. We introduce a pragmatic notion of net direction: "up" and "down" the net, "from one end to the other". This is enough to give a hint at the re-engineered domain.

Re-engineering Domain Operations

We first briefly sketch the tollgate Operations. Vehicles enter and leave the tollway net only at entry/exit hubs (toll plazas). Vehicles collect and return their tickets from and to tollgate ticket issuing, respectively payment machines. Tollgate ticket-issuing machines respond to sensor pressure from "passing" vehicles or by vehicle drivers pressing ticket-issuing machine buttons. Tollgate payment machines accept credit cards, bank notes or coins in designated currencies as payment and returns any change.

We then briefly introduce and sketch an operation performed when vehicles cross intersections: The vehicle is assumed to possess the ticket issued upon entry (in)to the net (at a tollgate). At the crossing of each intersection, by a vehicle, its ticket is sensed and is updated with the fact that the vehicle crossed the intersection.

The updated domain description section on support technology will detail the exact workings of these tollgate and internal intersection machines and the domain description section on human behaviour will likewise explore the man/machine facet.

Re-engineering Domain Events

The intersections are highway-engineered in such a way as to deter vehicle entry into opposite direction tollway links, yet, one never knows, there might still be (what we shall call ghost) vehicles, that is vehicles which have somehow defied the best intentions, and are observed moving along a tollway link in the wrong direction.

Re-engineering Domain Behaviours

The intended behaviour of a vehicle of the tollway is to enter at an entry hub (collecting a ticket at the toll gate), to move to the associated intersection, to move into, where relevant, either an upward or a downward tollway link, to proceed (i.e., move) along a sequence of one or more tollway links via connecting intersections, until turning into an exit link and leaving the net at an exit hub (toll plaza) while paying the toll.

• • •

This should be enough of a BPR rough sketch for us to meaningfully proceed to requirements prescription proper.

1.4.3 Domain Requirements Prescription

A domain requirements prescription is that part of the overall requirements prescription which can be expressed solely using terms from the domain description. Thus to construct the domain requirements prescription all we need is collaboration with the requirements stakeholders (who, with the requirements engineers, developed the BPR) and the possibly rewritten (resulting) domain description.

Domain Projection

By a *domain projection* we mean a *subset of the domain description, one which leaves out all those entities, functions, events, and (thus) behaviours that the stakeholders do not wish represented by the machine.*

The resulting document is a *partial domain requirements prescription*.

Domain Projection — Narrative.

We copy the domain description and call the copy a 0th version domain requirements prescription. From that document we remove all mention of link insertion and removal functions, to obtain a 1st version domain requirements prescription.

Domain Projection — Formalisation.

We do not show the resulting formalisation.

Domain Instantiation

By *domain instantiation* we mean a *refinement of the partial domain requirements prescription, resulting from the projection step, in which the refinements aim at rendering the entities, functions, events, and (thus) behaviours*

of the partial domain requirements prescription more concrete, more specific. Instantiations usually render these concepts less general.

Domain Instantiation — Narrative.

The 1st version domain requirements prescription is now updated with respect to the properties of the toll way net: We refer to Fig. 1.1 and the preliminary description given in Sect. 1.4.2. There are three kinds of hubs: tollgate hubs and intersection hubs: terminal intersection hubs and proper, intermediate intersection hubs. Tollgate hubs have one connecting two way link. linking the tollgate hub to its associated intersection hub. Terminal intersection hubs have three connecting links: (i) one, a two-way link, to a tollgate hub, (ii) one one-way link emanating to a next up (or down) intersection hub, and (iii) one one-way link incident upon this hub from a next up (or down) intersection hub. Proper intersection hubs have five connecting links: one, a two way link, to a tollgate hub, two one way links emanating to next up and down intersection hubs, and two one way links incident upon this hub from next up and down intersection hub. (Much more need be narrated.) As a result we obtain a 2nd version domain requirements prescription.

Domain Instantiation — Formalisation, Toll Way Net.

```
type
  TN = ((H × L) × (H × L × L))* × H × (L × H)
value
  abs_N: TN → N
  abs_N(tn) ≡ (tn_hubs(tn),tn_hubs(tn))
  pre wf_TN(tn)

  tn_hubs: TN → H-set,
  tn_hubs(hll,h,(_,hn)) ≡
    {h,hn} ∪ {thj,hj|((thj,tlj),(hj,lj,lj′)):
      ((H×L)×(H×L×L))•((thj,tlj),(hj,lj,lj′))∈ elems hlll}
  tn_links: TN → L-set
  tn_links(hll,_,(ln,_)) ≡ ... as above ...

theorem ∀ tn:TN • wf_TN(tn) ⇒ wf_N(abs_N(tn))
```

Domain Instantiation — Formalisation, Well-formedness.

```
type
  LnkM == plaza | way
```

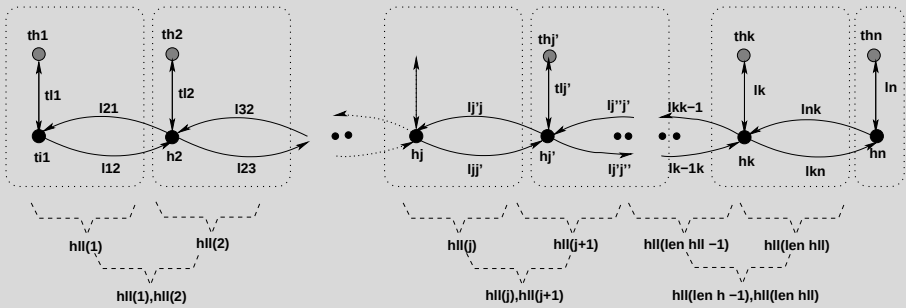

Fig. 1.2. A simple, linear toll road net: th_i: *t*oll *p*laza i, h_1, h_n: *t*erminal *i*ntersections, h_2, h_j, h'_j, h_k: *i*ntermediate *i*ntersections, $1<j\leq k$, $k=n$-1 l_{xy}, l_{yx}: toll-way *l*ink from h_x to h_y and from h_y to h_x, $1\leq x<n$. l_{x-1x}, l_{xx-1}: tollway *l*ink from h_{x-1} to h_x and h_x to h_{x-1}, $1\leq x<n$, dashed links are not in formulas.

```
value
  wf_TN: TN → Bool
  wf_TN(tn:(hll,h,(ln,hn))) ≡
    wf_Toll_Lnk(h,ln,hn)(plaza) ∧ wf_Toll_Ways(hll,h) ∧
    wf_State_Spaces(tn) [ to be defined under Determination ]

value
  wf_Toll_Ways: ((H×L)×(H×L×L))* × H → Bool
  wf_Toll_Ways(hll,h) ≡
    ∀ j:Nat • {j,j+1}⊆inds hll ⇒
      let ((thj,tlj),(hj,ljj′,lj′j)) = hll(j),
          (_,(hj′,_,_)) = hll(j+1) in
      wf_Toll_Lnk(thj,tlj,hj)(plaza) ∧
      wf_Toll_Lnk(hj,ljj′,hj′)(way) ∧ wf_Toll_Lnk(hj′,lj′j,hj)(way) end ∧
    let ((thk,tlk),(hk,lk,lk′)) = hll(len hll) in
      wf_Toll_Lnk(thk,tlk,hk)(plaza) ∧
      wf_Toll_Lnk(hk,lk,hk′)(way) ∧ wf_Toll_Lnk(hk′,lk′,hk)(way)  end

value
  wf_Toll_Lnk: (H×L×H) → LnkM → Bool
  wf_Toll_Lnk(h,l,h′)(m) ≡
    obs_Ps(l)={(obs_HI(h),obs_LI(l),obs_HI(h′)),
               (obs_HI(h′),obs_LI(l),obs_HI(h))} ∧
    obs_Σ(l) = case m of
                 plaza → obs_Ps(l),
                 way → {(obs_HI(h),obs_LI(l),obs_HI(h′))} end
```

Domain Determination

By *domain determination* we mean *a refinement of the partial domain requirements prescription, resulting from the instantiation step, in which the refinements aim at rendering the entities, functions, events, and (thus) behaviours of the partial domain requirements prescription less non-determinate, more determinate. Instantiations usually render these concepts less general.*

Domain Determination — Narrative.

We single out only two 'determinations': *The link state spaces.* There is only one link state: the set of all paths through the link, thus any link state space is the singleton set of its only link state. *The hub state spaces* are the singleton sets of the "current" hub states which allow these crossings: (i) from terminal link back to terminal link, (ii) from terminal link to emanating tollway link, (iii) from incident tollway link to terminal link, and (iv) from incident tollway link to emanating tollway link. Special provision must be made for expressing the entering from the outside and leaving toll plazas to the outside.

Domain Determination — Formalisation.

```
wf_State_Spaces: TN → Bool
wf_State_Spaces(hll,hn,(thn,tln)) ≡
  let ((th1,tl1),(h1,l12,l21)) = hll(1),
      ((thk,ljk),(hk,lkn,lnk)) = hll(len hll) in
  wf_Plaza(th1,tl1,h1) ∧ wf_Plaza(thn,tln,hn) ∧
  wf_End(h1,tl1,l12,l21,h2) ∧ wf_End(hk,tln,lkn,lnk,hn) ∧
  ∀ j:Nat • {j,j+1,j+2}⊆inds hll ⇒
    let (,(hj,ljj,lj′j)) = hll(j),((thj′,tlj′),(hj′,ljj′,lj′j′)) = hll(j+1) in
    wf_Plaza(thj′,tlj′,hj′) ∧ wf_Interm(ljj,lj′j,hj′,tlj′,ljj′,lj′j′) end end


wf_Plaza(th,tl,h) ≡
  obs_HΣ(th) = { [crossings at toll plazas]
      (″external″,obs_HI(th),obs_LI(tl)),
          (obs_LI(tl),obs_HI(th),″external″),
      (obs_LI(tl),obs_HI(th),obs_LI(tl))} ∧
  obs_HΩ(th) = {obs_HΣ(th)} ∧ obs_LΩ(tl) = {obs_LΣ(tl)}

wf_End(h,tl,l,l′) ≡
  obs_HΣ(h) = {[crossings at 3−link end hubs]
    (obs_LI(tl),obs_HI(h),obs_LI(tl)),(obs_LI(tl),obs_HI(h),obs_LI(l)),
    (obs_LI(l′),obs_HI(h),obs_LI(tl)),(obs_LI(l′),obs_HI(h),obs_LI(l))} ∧
  obs_HΩ(h) = {obs_HΣ(h)} ∧
  obs_LΩ(l) = {obs_LΣ(l)} ∧ obs_LΩ(l′) = {obs_LΣ(l′)}
```

wf_Interm(ul_1,dl_1,h,tl,ul,dl) ≡
  obs_HΣ(h) = {[crossings at properly intermediate, 5−link hubs]
    (obs_LI(tl),obs_HI(h),obs_LI(tl)),(obs_LI(tl),obs_HI(h),obs_LI(dl_1)),
    (obs_LI(tl),obs_HI(h),obs_LI(ul)),(obs_LI(ul_1),obs_HI(h),obs_LI(tl)),
    (obs_LI(ul_1),obs_HI(h),obs_LI(ul)),(obs_LI(ul_1),obs_HI(h),obs_LI(dl_1)),
    (obs_LI(dl),obs_HI(h),obs_LI(tl)),(obs_LI(dl),obs_HI(h),obs_LI(dl_1)),
    (obs_LI(dl),obs_HI(h),obs_LI(ul))} ∧
    obs_HΩ(h) = {obs_HΣ(h)} ∧ obs_LΩ(tl) = {obs_LΣ(tl)} ∧
    obs_LΩ(ul) = {obs_LΣ(ul)} ∧ obs_LΩ(dl) = {obs_LΣ(dl)}

Not all determinism issues above have been fully explained. But for now we should — in principle — be satisfied.

Domain Extension

By domain extension we understand the *introduction of domain entities, functions, events and behaviours that were not feasible in the original domain, but for which, with computing and communication, there is the possibility of feasible implementations, and such that what is introduced become part of the emerging domain requirements prescription.*

Domain Extension — Narrative.

The domain extension is that of the controlled access of vehicles to and departure from the toll road net: the entry to (and departure from) tollgates from (respectively to) an `"an external"` net — which we do not describe; the new entities of tollgates with all their machinery; the user/machine functions: upon entry: driver pressing entry button, tollgate delivering ticket; upon exit: driver presenting ticket, tollgate requesting payment, driver providing payment, etc.

One added (extended) domain requirements: as vehicles are allowed to cruise the entire net payment is a function of the totality of links traversed, possibly multiple times. This requires, in our case, that tickets be made such as to be sensed somewhat remotely, and that intersections be equipped with sensors which can record and transmit information about vehicle intersection crossings. (When exiting the tollgate machine can then access the exiting vehicles sequence of intersection crossings — based on which a payment fee calculation can be done.)

All this to be described in detail — including all the thinks that can go wrong (in the domain) and how drivers and tollgates are expected to react.

Domain Extension — Formalisation.

We suggest only some signatures:

```
type
  Mach, Ticket, Cash, Payment, Map_TN
value
  obs_Cash: Mach → Cash, obs_Tickets: M → Ticket-set
  obs_Entry, obs_Exit: Ticket → HI, obs_Ticket: V → (Ticket|nil)

  calculate_Payment: (HI×HI) → Map_TN → Payment

  press_Entry: M → M × Ticket              [gate up]
  press_Exit: M × Ticket → M × Payment
  payment: M × Payment → M × Cash          [gate up]
```

Domain Extension — Formalisation of Support Technology.

This example provides a classical requirements engineering setting for embedded, safety critical, real-time systems, requiring, ultimately, the techniques and tools of such things as Petri nets, statecharts, message sequence charts or live sequence charts and temporal logics (DC, TLA+).

Requirements Fitting

The issue of requirements fitting arises when two or more software development projects are based on what appears to be the same domain. The problem then is to harmonise the two or more software development projects by harmonising, if not too late, their requirements developments.

We thus assume that there are n domain requirements developments, d_{r_1}, d_{r_2}, ..., d_{r_n}, being considered, and that these pertain to the same domain — and can hence be assumed covered by a same domain description.

By requirements fitting we mean *a harmonisation of $n > 1$ domain requirements that have overlapping (common) not always consistent parts and which results in n 'modified and partial domain requirements', and m 'common domain requirements' that "fit into" two or more of the 'modified and partial domain requirements'.*

Requirements Fitting — Narrative.

We postulate two domain requirements: We have outlined a domain requirements development for software support for a toll road system. We have earlier hinted at domain operations related to insertion of new and removal of existing links and hubs. We can therefore postulate that there are two domain requirements developments, both based on the transport domain: one, $d_{r_{\text{toll}}}$, for a toll road computing system monitoring and controlling vehicle flow in

and out of toll plazas, and another, $d_{r_{\text{maint.}}}$, for a toll link and intersection (i.e., hub) building and maintenance system monitoring and controlling link and hub quality and for development.

The fitting procedure now identifies the shared of awareness of the net by both $d_{r_{\text{toll}}}$ and $d_{r_{\text{maint.}}}$ of nets (N), hubs (H) and links (L). We conclude from this that we can single out a common requirements for software that manages net, hubs and links. Such software requirements basically amounts to requirements for a database system. A suitable such system, say a relational database management system, DB_{rel}, may already be available with the customer.

In any case, where there before were two requirements $(d_{r_{\text{toll}}}, d_{r_{\text{maint.}}})$ there are now four: (i) $d'_{r_{\text{toll}}}$, a modification of $d_{r_{\text{toll}}}$ which omits the description parts pertaining to the net; (ii) $d'_{r_{\text{maint.}}}$, a modification of $d_{r_{\text{maint.}}}$ which likewise omits the description parts pertaining to the net; (iii) $d_{r_{\text{net}}}$, which contains what was basically omitted in $d'_{r_{\text{toll}}}$ and $d'_{r_{\text{maint.}}}$; and (iv) $d_{r_{\text{db:i/f}}}$ (for database interface) which prescribes a mapping between type names of $d_{r_{\text{net}}}$ and relation and attribute names of DB_{rel}.

Much more can and should be said, but this suffices as an example in a software engineering methodology paper.

Requirements Fitting — Formalisation.

We omit lengthy formalisation.

Domain Requirements Consolidation

After projection, instantiation, determination, extension and fitting, it is time to review, consolidate and possibly restructure (including re-specify) the domain requirements prescription before the next stage of requirements development.

1.5 Discussion

1.5.1 An 'Odyssey'

Our 'Odyssey' has ended. A long example has been given.

We have shown that requirements engineering can have an abstraction basis in domain engineering; and we have shown that we do not have to start software development with requirements engineering, but that we can start software development with domain engineering and then proceed to a more orderly requirements engineering phase than witnessed today.

1.5.2 Claims of Contribution

What is essentially new here is the claim and its partial validation that one can and probably should put far more emphasis on domain modelling, the domain modelling concepts, principles and techniques of business process domain intrinsics, domain support technologies, domain management and organisation, domain rules and regulations, domain scripts and domain human behaviour; the identification of, and the decomposition of the requirements development process into, domain requirements, interface requirements and machine requirements; the domain requirements "derivation" concepts, principles and techniques of projection, instantiation, determination, extension and fitting and the identification of structuring of the interface ground requirements shared entities, shared operations, shared events and shared behaviours.

1.5.3 Comparison to Other Work

Jackson's Problem Frame approach [4] cleverly alternates between domain analysis, requirements development and software design. For more satisfactory comparisons between our domain engineering approach and past practices and writings on domain analysis we refer to [3].

1.5.4 A Critique

A major presentation of domain and of requirements engineering is given in [1, Chaps. 8–16 and 17–24]. [3] provides a summary, more complete presentation of domain engineering than the present paper allows, while [2] discusses a set of research issues for domain engineering. Papers, like [3, 2], but for requirements engineering, with more a complete presentation, respectively a discussion of research issues for this new kind of requirements engineering might be desirable. The current paper's Sect. 1.4 provided a slightly revised structuring of the interface requirements engineering.

Some of the development steps within the domain modelling and likewise within the requirements modelling are refinements, and some are extensions. If we ensure that the extensions are what is known as Conservative extensions then all theorems of the source of the extension go through and are also valid in the extension. Although such things are here rather clear much more should be said here about ensuring Conservative extensions. We do not since the current paper is is not aimed at the finer issues of the development but at the domain to requirements "derivation" issues.

1.5.5 Programming Methodology versus Software Engineering

The following question has been formulated: *How to make a programming methodological approach like this become everyday software engineering practice ? For example, a small company willing to launch on the market a new*

idea in a specific domain, needs under this approach to build up a full domain formalization. I fear that this could be felt as a too large burden. On the other hand, using pre-cooked, public, standardized or third-party formalizations of specific domains could end in constraining the imagination of innovators ?

The programming methodological answer is: Yes, one must build a domain description, informal and, ideally speaking also formal.

The software engineering answer is: how "full" it should be: at least "big" enough to encompass the requirements.

The science and engineering answer is: public universities must experimentally develop and research sufficiently broad (scope) domain theories, while private software houses adapt these to their narrower (span) application domains, thus establishing proprietary, corporate assets.

The research answer is: We must study a programming methodology like the one put forward in this paper. We must do so because the programming methodology appears logical, sound. We cannot abstain from studying this programming methodology just because (even a majority of) software engineers "feels" that it *is too large a burden* to follow this approach "slavishly".

1.6 Acknowledgments

I gratefully acknowledge support from Université Henri Poincaré (UHP), Nancy, and from INRIA (l'Institut National de Recherche en Informatique et en Automatique) both of France, for my two month stay at LORIA (Laboratoire Lorrain de Recherche en Informatique et ses Applications), Nancy, in the fall of 2007. I especially and warmly thank Dominique Méry for hosting me. And I thank the organisers of Ugo Montanari's Festschrift, Pierpaolo Degano, Jose Meseguer and Rocco De Nicola, for inviting me — thus forcing me to willingly write this paper.

References

1. Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design.* Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
2. Dines Bjørner. Domain Theory: Practice and Theories, Discussion of Possible Research Topics. In *ICTAC'2007*, volume 4701 of *Lecture Notes in Computer Science (eds. J.C.P. Woodcock et al.)*, pages 1–17, Heidelberg, September 2007. Springer.
3. Dines Bjørner. Domain Engineering. In *BCS FACS Seminars*, Lecture Notes in Computer Science, the BCS FAC Series (eds. Paul Boca and Jonathan Bowen), pages 1–42, London, UK, 2008. Springer. To appear.
4. Michael A. Jackson. *Problem Frames — Analyzing and Structuring Software Development Problems.* ACM Press, Pearson Education. Addison–Wesley, Edinburgh Gate, Harlow CM20 2JE, England, 2001.